Department of Information Systems
Subject : Requirement Engineering

Software Engineering

IS184309, 3 sks

Dr. Rarasmaya Indraswari, S.Kom.

Outline

- ✓Recap: What makes project fail?
- ✓Software Engineering
- ✓Software Engineering Body of Knowledge (SWEBOK)
- ✓ Software Process/Software Development Life Cycle (SDLC)
- ✓Software Myths

What makes project fail?

Problems of scope

The boundary of the system may be ill-defined

- Who is going to interact with the system?
- What other systems are involved?
- Exactly what functionality is the responsibility of the system
  - e.g. should a rostering system produce a telephone directory?

The customer/user may specify unnecessary technical detail that may confuse overall system objectives

- e.g. specifying OS, language, hardware, etc. for no particularly good reason

Problems of Understanding

Customers/users may:

- Not be completely sure of what is needed, e.g.:
  - "See what you can do to help us" (Marketing director of textile business)
  - "Try to improve the project" (Director of British Aircraft Corporation, with reference to the Concorde project)
- Have a poor understanding of the capabilities and limitations of their computing equipment
- Not have a full understanding of the problem domain
- Have trouble communicating needs to system engineer
- Omit information believed to be "obvious"
- Specify requirements that conflict with needs of others
- Specify requirements that are ambiguous or untestable

Problem of volatility

Requirements change over time

Change is inevitable in most systems, due to factors such as:

- Changes in customer organization, e.g. new divisions, new products
- Changes in scale, e.g.
  - Number of transactions per day
  - Number of users
  - Increased connection bandwidth
- External changes
  - Changes in law (e.g. taxation)
  - Changes in international standards (e.g. MPEG)
- Customers get new ideas as they become aware of system possibilities

Short tips

Aim is to build the right system

- A system that meets the user's needs

Requires the collaboration of several groups of participants with different backgrounds

- Knowledge GAP

The software engineer needs to :

- Learn about application domain and "discover" requirements
- Choose an appropriate representation for specifying requirements

Short tips (definition of application domain)

Application domain is **an organization that administrates, monitors, or controls a problem domain** (Mathiassen, 2000)

- Objective: to analyze the user's requirements
- In application domain, there are 3 components: **Usage, Function, Interface**
- **Usage** refers to how system interacts with the actors
  - **Actor** is an abstraction of users or other system that interact with the target systems
- **Function** is a facility for making a model useful for actors
- **Interface** or **User Interface** is facilities that make a system's model and function available to actors

Software engineer needs to learn the application domain to discover the requirements

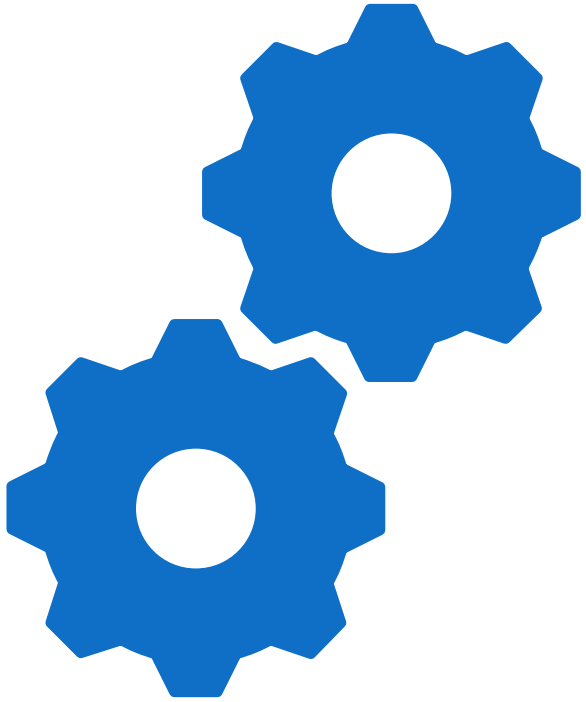
- collect data on application domain (may include existing system)
  - investigation → existing documentation
  - observation → work practices
  - interviews → questionnaires, personal
  - prototyping → interface, functions

Need effective communication skills

- challenge client's/users' model of the world
  - people: a) filter out, b) distort, and/or c) generalize information
    - ➥ e.g., challenge statements containing universal quantifiers such as, all, everyone, always, never, nobody, none!
- elicit problems, not solutions
  - **Not this:** "We need to put in a teleconferencing network among our offices."
  - **But this:** "Our offices are not communicating in a timely manner."
- distinguish needs from wants; prioritize needs
  - needs: features critical to the system's success
  - wants: features nice to have, but not essential

If you can't describe what you are doing as a process, you don't know what you're doing.

(William Edwards Deming, management consultant, 1900–93)



Software Engineering

Intro

Software is like a werewolf—it looks normal until the moon comes out and it turns into a monster

- Missed deadlines
- Blown budgets
- Buggy software

We want the silver bullet to kill the monster

- something to make software costs drop as rapidly as computer hardware costs do.

Software Crisis

- Therac-25 Radiation,
- Ariane Explosion,
- Denver International Airport,
- NORAD,
- Chinook Helicopter,
- NASA Mars Climate Orbiter,
- etc.

Software Crisis

- “Difficulty of writing useful and efficient computer programs in the required time”
- Due to the rapid increases in computer power and the complexity of the problems that could not be tackled → existing method become inadequate
- Causes:
  - Projects running over-budget
  - Projects running over-time
  - Software was very inefficient
  - Software was of low quality
  - Software often did not meet requirements
  - Projects were unmanageable and code difficult to maintain
  - Software was never delivered

Software is …

The product that software professionals build and then support over the long term. A logical rather than a physical system element

Some Definition

- **Instructions** → executed provide desired features, function, and performance
- **Data structures** → enable the programs to adequately manipulate information
- **Documents** → describe the operation and use of the programs

Software Dual Role

- As a product
  - Transforms information - produces, manages, acquires, modifies, displays, or transmits information
  - Delivers computing potential of hardware and networks
- As the vehicle for delivering product
  - control other programs (operating systems),
  - communication of information (networks), and
  - the creation and control of other programs (software tools and environments)

Software Categories

Embedded Software

Resides in **within** a product or system. Used to **implement** and **control** features and functions for the end-user and for the system itself. Can perform limited and esoteric functions.

Ex. keypad control for a microwave oven.

Product Line Software

Designed to provide a specific capability for use by many different customers, able to focus on a limited and esoteric marketplace, able to address mass consumer markets

Ex. Word processing, spreadsheet, CG, multimedia, etc.

Software Categories

Web Applications

Can be little more than a set of linked hypertext files.

It evolving into sophisticated computing environments that not only provide standalone features, functions but also **integrated** with corporate database and business applications.

Ex. Integra, Share ITS

Artificial Intelligence software

Makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis

Ex. Robotics, expert system, game playing, etc.

Software—New Categories

Ubiquitous Computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to **communicate across vast network**.

Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.

Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)

What is Software Engineering?

A BRIEF HISTORY OF SOFTWARE ENGINEERING

According to the IEEE **Software Engineering** means applying the principles of engineering to the software development field.

1968

Computers became available, first for some people and then for the rest.

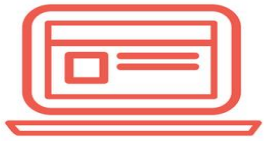
As their popularity grew also did their capacity and the complexity of the **software** they needed.

1968

The **NATO Conference** took place and people came up with the concept of **Software Engineering**.

They thought that software was in crisis and they needed a **new discipline** that focused in making software less expensive and more effective.

Software crisis remained but people like **Dijkstra** and **Hoare** offered smart and different solutions.



1975

Microcomputers first appeared on the market, they were based on single-chip processors with 8-bit data buses and 32 Kbytes or less of memory.

These advances made computers **affordable for individuals.**

Unix was released and people liked it because it was perfect for the rapidly emerging minicomputers.

The concept of **modularization**, was born.

Object Oriented Programming surged as the need for more complex software increased.

Nowadays **software engineers** have a lot of **resources** and **techniques** they can use to make software projects, anyways they still **struggle** to make low cost, great quality, not time consuming and effective software.

Why Software Engineering?

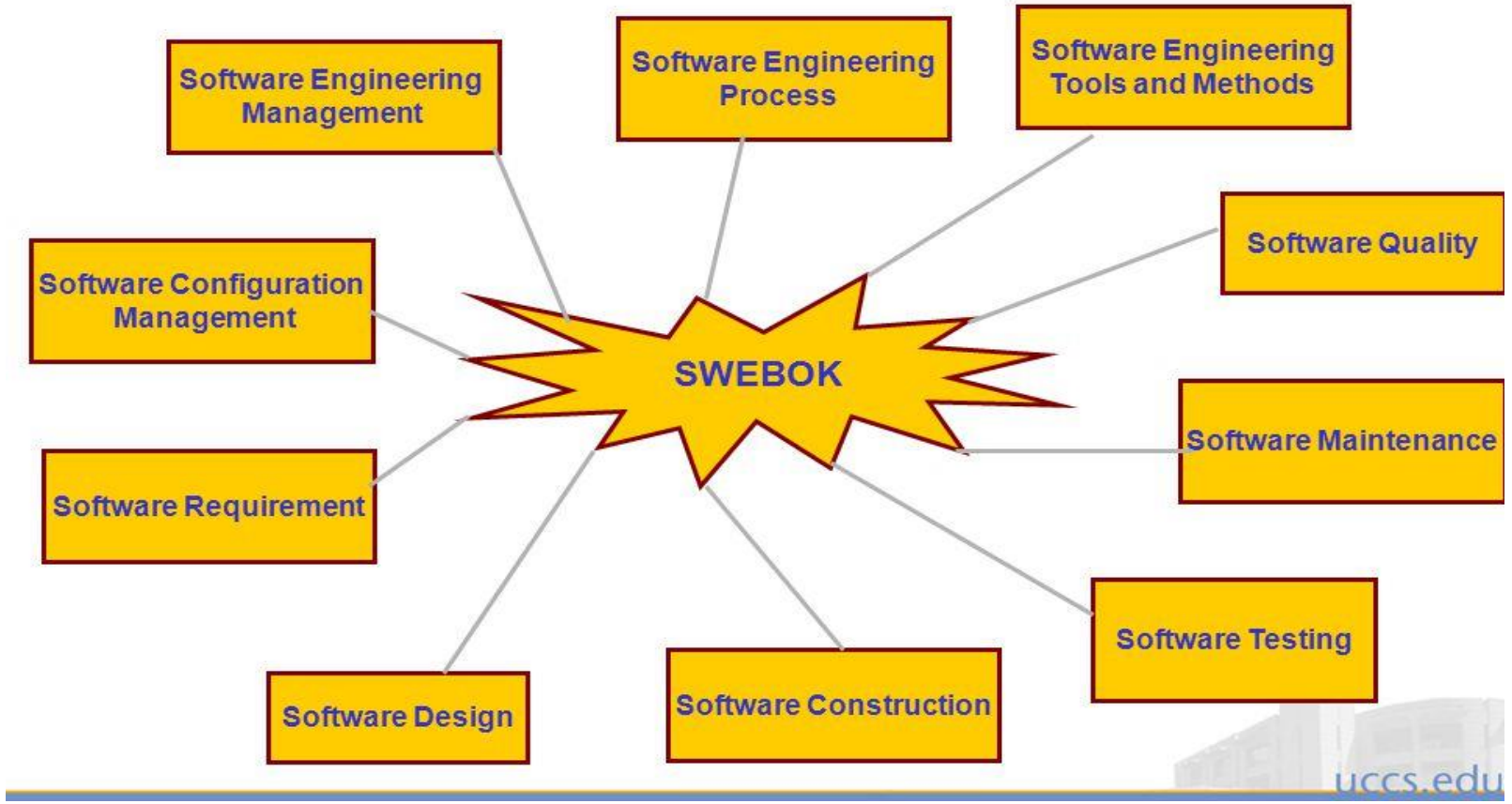
A naive view:

Problem Specification *coding*  Final Program

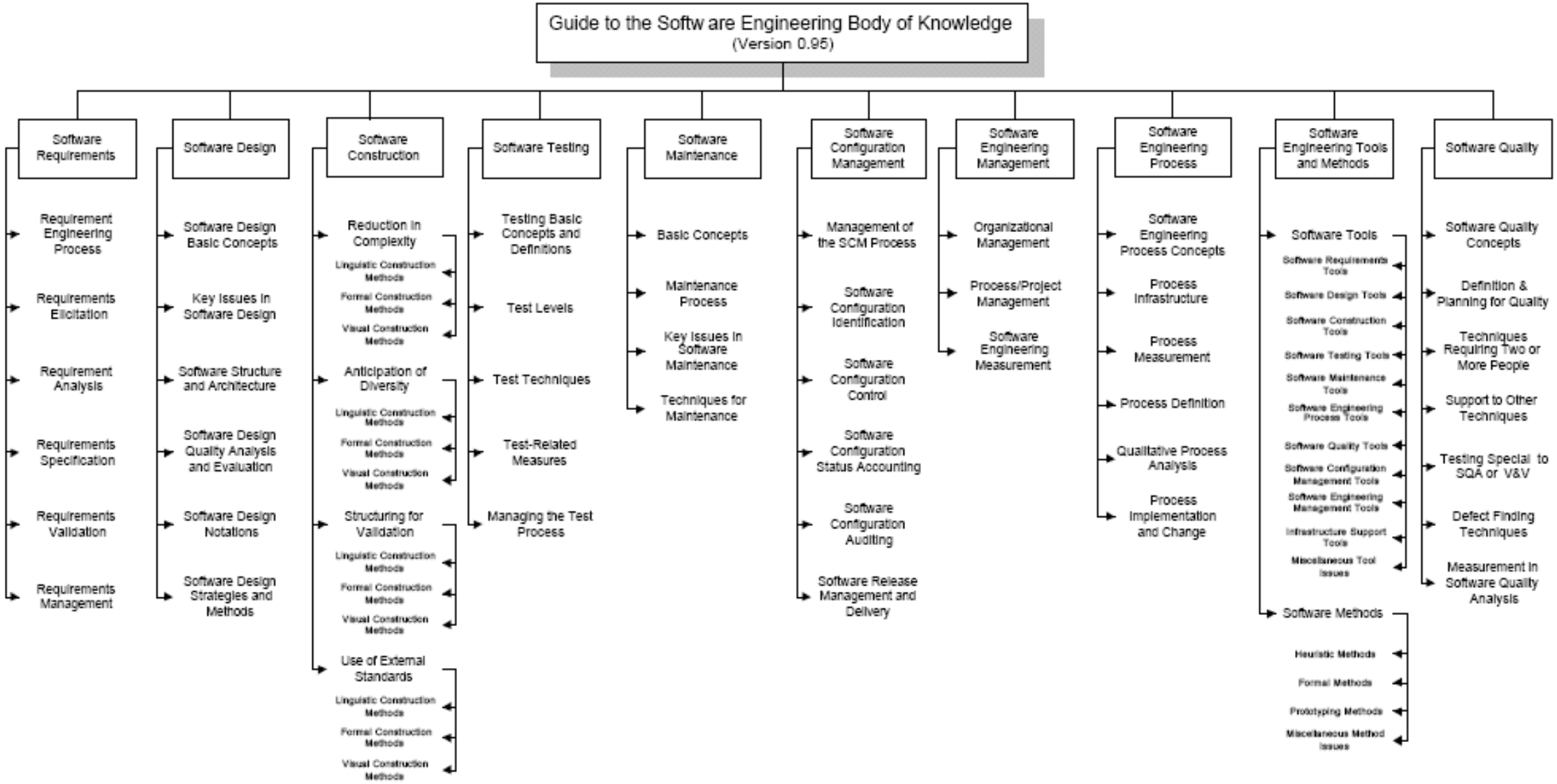
But ...

- Where did the *specification* come from?
- How do you know the specification corresponds to the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the users' *needs change*?
- How do you *divide tasks up* if you have more than a one-person team?

A complete SWEBOK based Education Program



SWEBOK


Guide to the Softw are Engineering Body of Knowledge
(Version 0.95)
Software Requirements
Requirement Engineering Process
Requirements Elicitation
Requirement Analysis
Requirements Specification
Requirements Validation
Requirements Management
Software Design
Software Design Basic Concepts
Key Issues in Software Design
Software Structure and Architecture
Software Design Quality Analysis and Evaluation
Software Design Notations
Software Design Strategies and Methods
Software Construction
Reduction in Complexity
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Anticipation of Diversity
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Structuring for Validation
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Use of External Standards
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Software Testing
Testing Basic Concepts and Definitions
Test Levels
Test Techniques
Test-Related Measures
Managing the Test Process
Software Maintenance
Basic Concepts
Maintenance Process
Key Issues in Software Maintenance
Techniques for Maintenance
Software Configuration Management
Management of the SCM Process
Software Configuration Identification
Software Configuration Control
Software Configuration Status Accounting
Software Configuration Auditing
Software Release Management and Delivery
Software Engineering Management
Organizational Management
Process/Project Management
Software Engineering Measurement
Software Engineering Process
Software Engineering Process Concepts
Process Infrastructure
Process Measurement
Process Definition
Qualitative Process Analysis
Process Implementation and Change
Software Engineering Tools and Methods
Software Tools
Software Requirements Tools
Software Design Tools
Software Construction Tools
Software Testing Tools
Software Maintenance Tools
Software Engineering Process Tools
Software Quality Tools
Software Configuration Management Tools
Software Engineering Management Tools
Infrastructure Support Tools
Miscellaneous Tool Issues
Software Methods
Heuristic Methods
Formal Methods
Prototyping Methods
Miscellaneous Method Issues
Software Quality
Software Quality Concepts
Definition & Planning for Quality
Techniques Requiring Two or More People
Support to Other Techniques
Testing Special to SQA or V&V
Defect Finding Techniques
Measurement in Software Quality Analysis

SWEBOK Certification

SWEBOK®

CERTIFICATE PROGRAM

Software Development Activities

Requirements Collection	Establish customer's needs
Analysis	Model and specify the requirements ("what")
Design	Model and specify a solution ("how")
Implementation	Construct a solution in software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to new requirements

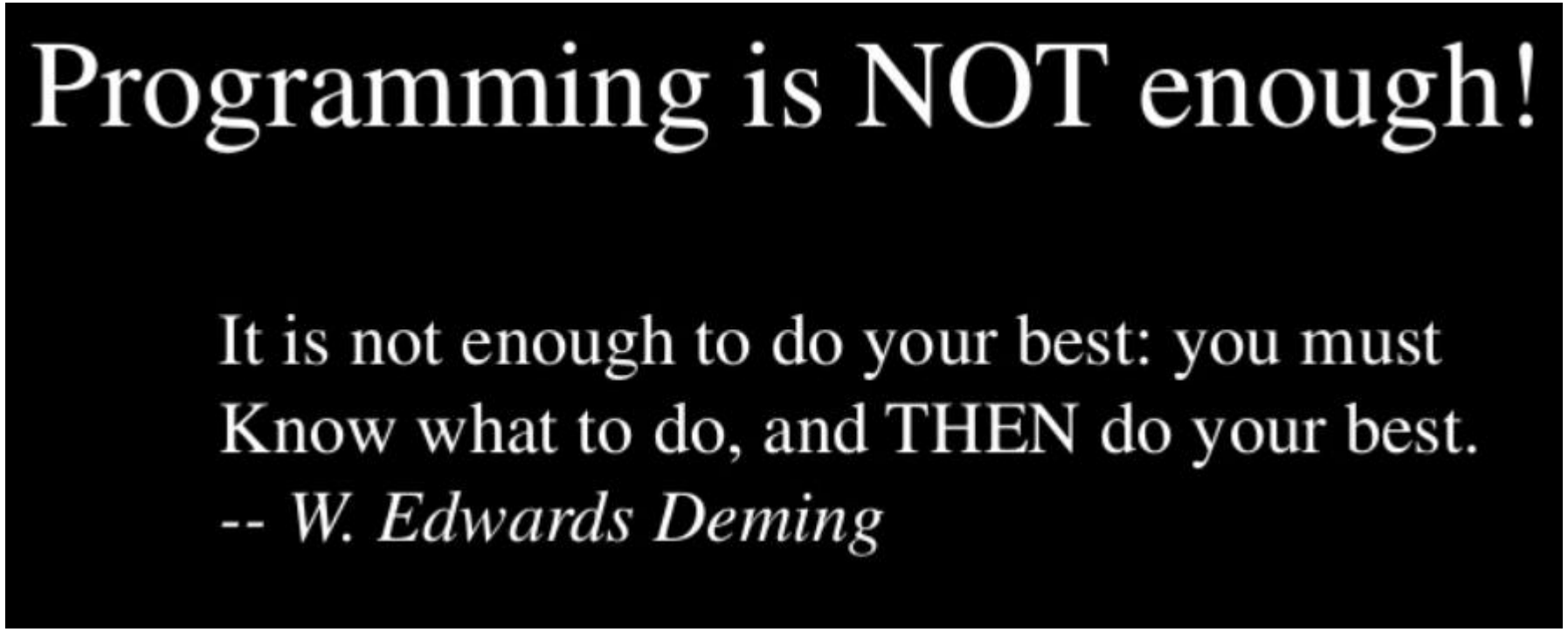
NB: these are ongoing activities, not sequential phases!

Team in Software Development Process

Software Process Model, SDLC

1) Incremental Process Model
   i. The Incremental Model
   ii. The RAD (Rapid Application Development) Model
2) Evolutionary Process Model
   i. Prototyping
   ii. The Spiral Model
3) Specialized Process Model
   i. Component-Based Development
   ii. The Formal Methods Model
4) Agile Process Model
   i. Extreme Programming (XP)
   ii. Scrum
5) Use Case Driven Object Modeling – Iconix Process
6) Unified Process
7) Waterfall & V-Model

Conclusion


Programming is NOT enough!
It is not enough to do your best: you must
Know what to do, and THEN do your best.
-- W. Edwards Deming

No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

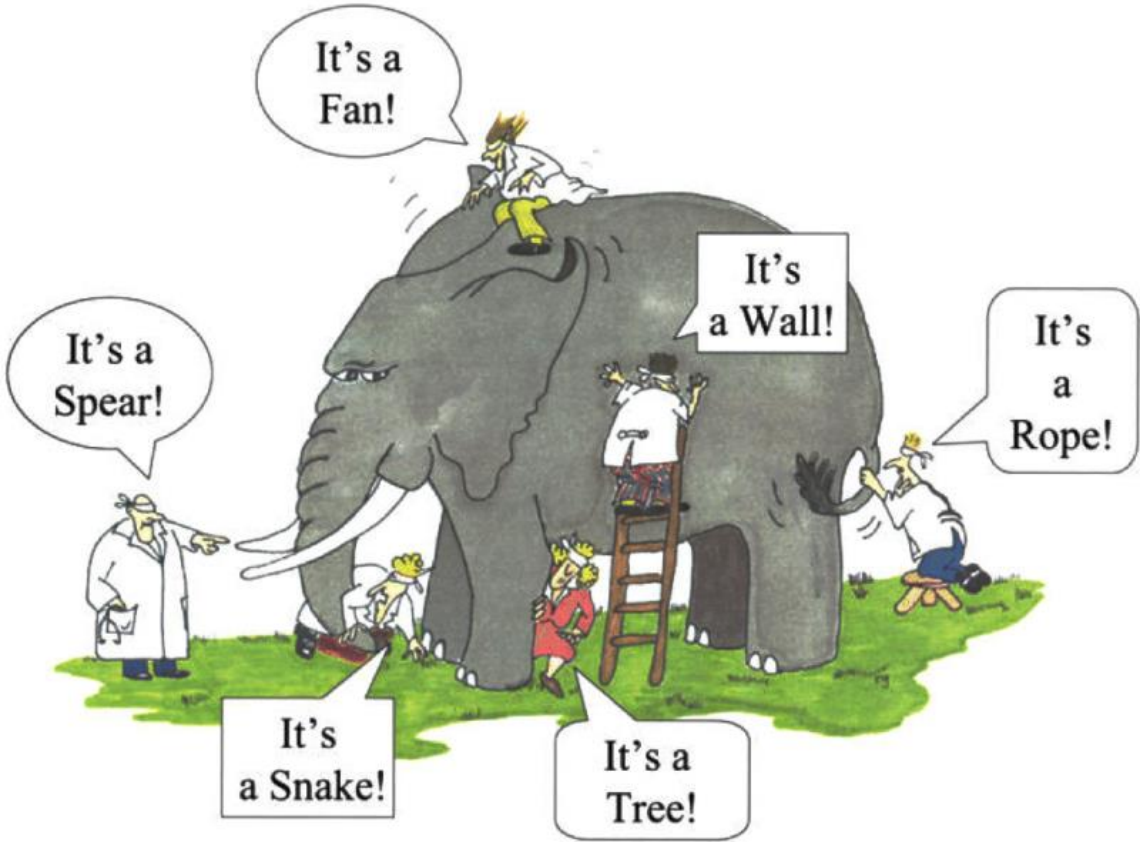
There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

References

- http://scg.unibe.ch (Institute of Computer Science (INF) in the Faculty of Science of the University of Bern)
- www.uccs.edu (Computer Science of University of Colorado Colorado Springs)
- http://www.semq.eu/leng/swebok.htm
- http://www.cse.psu.edu/~gxt29/bug/softwarebug.html
- *Software Engineering*. Ian Sommerville. Addison-Wesley Pub Co; ISBN: 020139815X, 7th edition, 2004
- *Software Engineering: A Practioner's Approach*. Roger S. Pressman. McGraw Hill Text; ISBN: 0072496681; 5th edition, 2001
- *No Silver Bullets, Essence and Accidents of Software Engineering*. Frederick P. Brooks, Jr., 1987, Devon Simmonds Computer Science Department University of North Carolina Wilmington

Software Myths

Software Myths



Myths vs Facts (reality)

Software Myths

Definition: Beliefs about software and the process used to build it.

Myths have number of **attributes** that have made them insidious (i.e. dangerous).

Misleading Attitudes - caused serious problem for managers and technical people.

Management Myths #1

Managers in most disciplines, are often under pressure to maintain budgets, keep schedules on time, and improve quality.

Myth1: We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?

Reality :

Are software practitioners aware of existence standards?

Does it reflect modern software engineering practice?

Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality?

Management Myths #2

Myth2: If we get behind schedule, we can add more programmers and catch up

Reality*:*

Software development is not a mechanistic process like manufacturing. Adding people to a late software project makes it later.

People can be added but only in a planned and well-coordinated manner

Management Myths #3

Myth3: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality*:*

If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsource software projects

Customer Myths

Customer may be a person from **inside** or **outside** the company that has requested software under contract

In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation

Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer

Customer Myths #1

Myth1: A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.

Reality*:*

A poor up-front definition is the major cause of failed software efforts.

A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential.

These characteristics can be determined only after thorough communication between customer and developer.

Customer Myths #2

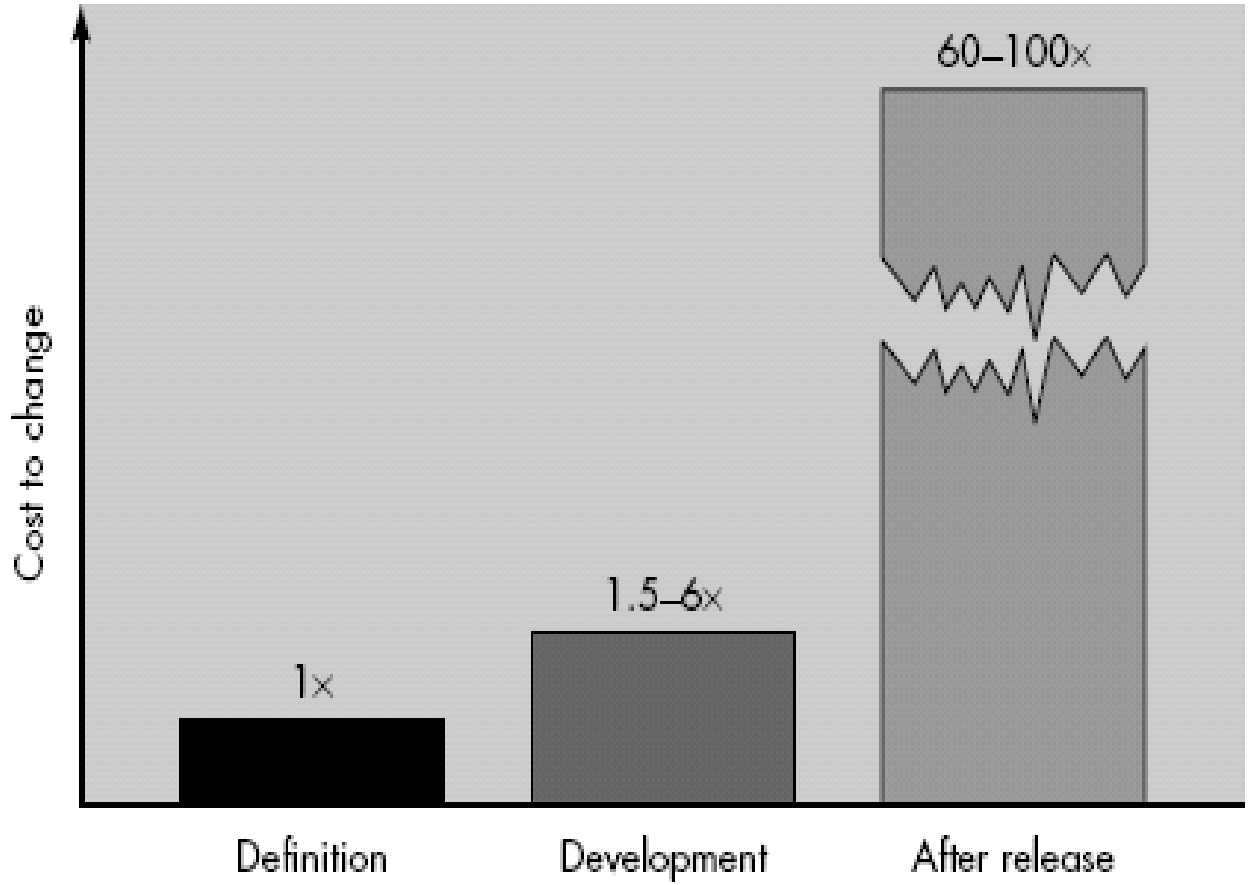
Myth2: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality*:*

Customer can review requirements and recommend modifications with relatively little impact on cost.

When changes are requested during software design, the cost impact grows rapidly.

Cost on Software Changes



Practitioner's Myths #1

Myth1: Once we write the program and get it to work, our job is done.

Reality:

Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done."

Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Practitioner's Myths #2

Myth2: Until I get the program "running" I have no way of assessing its quality.

Reality:

One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review.

Software reviews are a "*quality filter*" that have been found to be more effective than testing for finding certain classes of software errors.

Practitioner's Myths #3

Myth3: The only deliverable work product for a successful project is the working program.

Reality:

A working program is only one part of a software configuration that includes many elements.

Documentation provides a foundation for successful engineering and, more important, guidance for software support

Practitioner's Myths #4

Myth3: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Reality:

Software engineering is **NOT** about creating documents.

It is **ABOUT** creating **QUALITY**.

Better quality leads to **reduced** rework.

And **reduced** rework results in **faster** delivery times.

Task 2

1. Create a group of 3-4 students for class B and 4-5 students for class A
2. There should be a total of 12 groups in a class
3. Each group should make a presentation (file only) regarding the corresponding topic number:
    1. Waterfall, V-Model
    2. Evolutionary development
    3. Formal systems development
    4. Reuse-oriented development
    5. Incremental development
    6. Extreme programming
    7. Spiral development
    8. Concurrent Model
    9. Unified Process
    10. Adaptive Software Development
    11. Lean Software Development
    12. Scrum
4. Submit your presentation file (.pptx) in the Classroom. The content of your presentation will be reviewed, and revision will be given if any.
5. The length / number of presentation slide is not limited.